NODE JS

PROGRAMAÇÃO WEB 1

OBJETIVOS DE APRENDIZAGEM

- Introduzir os conceitos iniciais de Node JS como ferramenta *web* no lado do servidor

AGENDA

- Por que Node JS?
- Arquitetura
- Instalação
- Primeiros passos
- Ferramentas
- Exercícios
- Próximos passos

“As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.”

nodejs.org

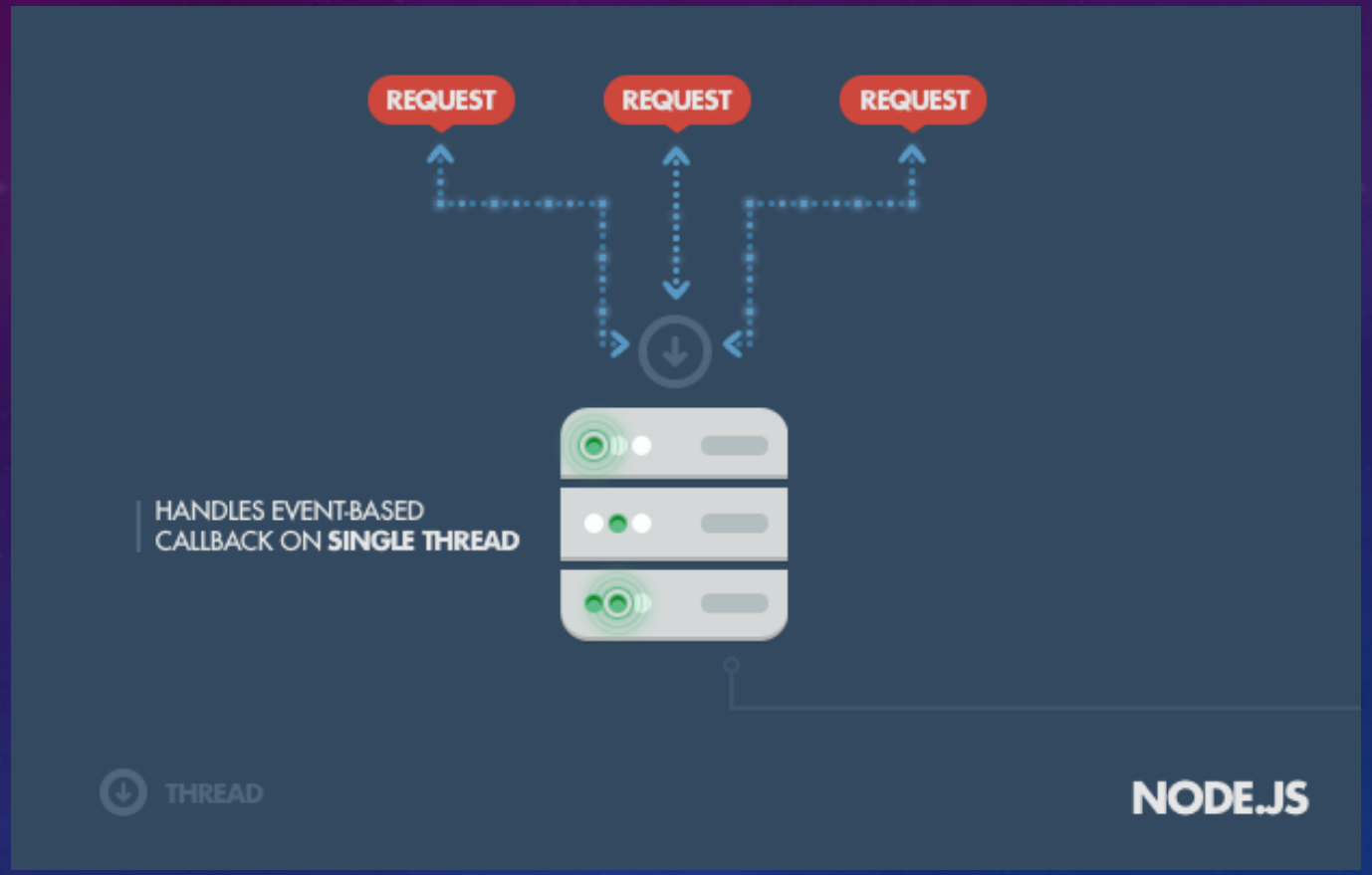
POR QUE NODE JS?

- Aplicações web escaláveis em tempo real para uma grande quantidade de conexões simultâneas
- Trata-se da implementação da implementação de uma engine JS fora do ambiente do navegador
- Foi idealizado para rodar aplicações em tempo real de alta escalabilidade na Internet
- Fundamenta-se na pilha HTML+CSS+JS

ARQUITETURA

- Node JS utiliza uma arquitetura não bloqueante baseada em eventos
- Bastante leve e eficiente para aplicações com uso intensivo de dados
- Abordagem distinta de servidores web tradicionais
- Visa a baixa utilização de recursos do servidor


REQUEST
REQUEST
REQUEST
REQUEST
REQUEST
SERVER CREATES NEW THREAD
FROM **LIMITED POOL**
OR **WAITS** FOR
AVAILABLE THREAD
THREAD
WAITING
TRADITIONAL


REQUEST
REQUEST
REQUEST
HANDLES EVENT-BASED
CALLBACK ON SINGLE THREAD
THREAD
NODE.JS

ESCALABILIDADE

- Estima-se que um webserver típico com 8Gb de memória seja capaz de atender 4000 conexões (8GB / 2Mb da *thread* cada conexão)
- Estima-se que um servidor Node JS seja capaz de lidar com 1M de conexões concorrentes

SINGLE THREAD

- A grande questão é o compartilhamento da única thread com as diversas requisições dos clientes
- O primeiro ponto fraco são aplicações de computação intensa
- O segundo é evitar que exceções e erros de execução congelem ou forcem a finalização da instância do Node JS

TÉCNICAS

- Passar erros como parâmetros de *callback* para o cliente (caller)
- Utilizar ferramentas de monitoramento específicas para o caso de exceções não tratadas
- Recuperação de instâncias

INSTALAÇÃO

1. https://nodejs.org/en/
2. Instalação Visual Studio Code + Node JS (https://www.youtube.com/watch?v=rxNY_mMZD_8)

PRIMEIROS PASSOS

- Uso do Node JS para desenvolvimento web
- Módulo http
  - Trata do atendimento a requisições HTTP, métodos e cabeçalhos

- Utilizando o console, executar comando node
- Digitar o código ao lado
- Acessar o navegador (http://127.0.0.1:3000)
- O que acontece?

```
const http = require('http');

const server = http.createServer((request, response) => {
  console.log('Servidor ativo....');

});

server.listen(3000)
```

- No exemplo anterior a função `http.createServer()` recebe uma função *callback* com dois objetos:
  - `request` que contém a requisição recebida do navegador
  - `response` que trata a resposta dada pelo servidor Node JS ao receber a requisição

- Modificando o exemplo para exibir informações da requisição no console. Ver ao lado
- Acessar o navegador (http://127.0.0.1:3000)
- O que acontece?

```
const http = require('http');
const server = http.createServer((request, response)=>{
    const headers = request.headers;
    const method = request.method;
    const url = request.url;
    console.log("Headers");
    console.log(headers);
    console.log("Method: " + method);
    console.log("URL: " + url);
});
server.listen(3000);
```

- Para desenvolvimento web é interessante que a saída seja o próprio navcegador e não o terminal como nos exemplos anteriores
- O código ao lado exemplifica isso
- Acessar o navegador (http://127.0.0.1:3000)
- O que acontece?

```
const http = require('http');
const server = http.createServer((request, response)=>{
    const method = request.method;
    const url = request.url;
    response.setHeader('Content-type', 'text/html');
    response.write('<h1>Dados da requisição</h1><br>'); response.write('<hr>');
    response.write('Método da requisição: ' + method + '<br>'); response.write('URL da requisição:' + url + '<br>');
    response.end()
});
server.listen(3000)
```

- Formatando um documento padrão HTML5
- O exemplo ao lado mostra a inclusão de um documento HTML completo
- Acessar o navegador (http://127.0.0.1:3000)
- O que acontece?

```
const http = require('http');

const server = http.createServer((request,
response)=>{

    const method = request.method;

    const url = request.url;

    response.setHeader('Content-type', 'text/html');

    response.write('<!DOCTYPE html><html lang="pt-br">');
    response.write('<head><meta charset="UTF-8"></head>');
    response.write('<body>')

    response.write('<h1>Dados da requisição</h1><br>');
    response.write('<hr>');

    response.write('Método da requisição: ' + method +
    '<br>'); response.write('URL da requisição:' + url +
    '<br>'); response.write('</body></html>');

    response.end()

});

server.listen(3000)
```

FERRAMENTAS

- Terminal
- NodeJS instalado
- IDE de preferência
  - Visual Studio Code
  - Web Storm

PRÓXIMOS PASSOS

- Conhecer módulos para incrementar o desenvolvimento
  - express.js
  - *Template engines*
  - Mongo DB, dentre outras soluções de Bancos de Dados

EXERCÍCIOS

1. Reproduzir os códigos anteriores usando a ferramenta de sua preferência (Linha de comando, Visual Studio Code, Replit)
2. Criar um index.js que implementa um servidor HTTP que fornece um HTML estático (Hello, world!!!)
3. Criar um index.js que exiba as propriedades da conexão, tais como:
    1. Método utilizado;
    2. URL
    3. Cabeçalho
    4. Formatar os dados acima (tabela, div ou similar)

REFERÊNCIAS

- https://nodejs.org/en/
- https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js